Implementation of the game 'Guess Who' using the GLIF Framework

Rithika Cariappa
M.Sc. Artificial Intelligence
FAU Erlangen-Nürnberg
rithika.cariappa@fau.de

Yutong Wu
M.Sc. Informatik
FAU Erlangen-Nürnberg
yutong.wu@fau.de

1 Introduction

With rapid growth and advance in computing technologies, Artificial Intelligence has been key in finding ways to beat opponents or minimize loss in logical and sometimes even in games that rely purely on chance. It dates back to 1956, when IBM presented their Checkers AI and won against a human player for the first time 6 years later. Since then, many programs have been developed with one of the most notable milestones being the game between AlphaGo and Lee Sedol in 2016 (Kurenkov, 2016). Many modern-day games are linguistic and/or symbolic in nature, such as Rock-Paper-Scissors, Tic-Tac-Toe, Hangman & Cluedo and serve as a good basis for logic-based game solving. In our project, we use one such game 'Guess Who' to demonstrate how we can leverage logic based natural language understanding for solving such games. Leveraging the GLIF Framework(Schaefer et al., 2019), we build a pipeline for processing user input with abstract and concrete grammar to accept and parse some natural language statements and do the semantic contruction of the grammar trees into logic with MMT . Then we build an inference model using ELPI to check if the characters have certain characteristics and given some features make a guess as to who the character is.

2 Background

'Guess Who' is a two-player board game developed by Ora and Theo Coster under the banner of Theora Designs and manufactured first in 1979 by Milton Bradley and is currently owned by Hasbro, Inc. The game consists of 2 boards with 24 opening and closing tabs of pictures of characters with their names, each with characteristic features. Each player has a board, the images facing themselves. The player then proceeds to choose a character of their choice and places it in front of themselves, away from the view of the opponent. They then take turns asking questions about the features or name of the opponent's chosen card to which the opponent can only answer with "Yes" or "No". The player to guess the opponent's character first wins(Hasbro, 1979).

The game was thought to have been best solved using the binary search method with O(log n) complexity, with options being eliminated based on the question and answers of features. The order of questions was assumed to even be optimised to this end to win (Rober, 2015). However, Nica (2016) showed that the optimal strategy for a player who is losing is to make certain "bold plays" and this could lead to a win.

GLIF is a Natural Language Understanding framework for logic-based projects, developed by Schaefer et al. (2019) using the Grammatical Framework(Ranta, 2011), Meta-Meta-Theory/Tool

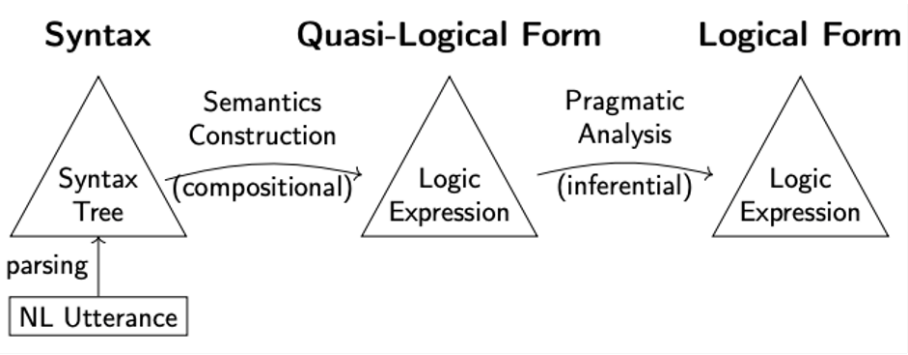

Figure 1: Natural Language Utterance to Logical Expression Pipeline (Kohlhase et al., 2023)

(Rabe and Kohlhase, 2013) and an Embedded λProlog Interpreter (Dunchev et al., 2015). GF can be used to accept user input as natural language fragments and either accept it as valid or invalid grammar based on the rules specified in the abstract and concrete grammar. From here, we can obtain grammar trees that can be passed to MMT for semantic construction: from grammar trees to a logic that is not human language dependent. It provides a way to represent the knowledge we have for computational purposes. MMT employs the Curry/Howard isomorphism and treats axioms/conjectures as typed symbol declarations (propositions-as-types), inference rules as function types (proof transformers) and theorems as definitions (proof terms for conjectures). ELPI can be run standalone for inference by defining types, propositions, truth values and rules. Altogether, the GLIF kernel provides a way to implement the entire pipeline from natural language strings to a logical expression and then do inference for applications such as ambiguity resolution and theorem proving.

3 Implementation

3.1 Characters and Characteristics

For this implementation we created a set of people and features:

- People
  - Mary, Peter, Fido, John
- Genders
  - Female, Male, Other
- Hair Colour
  - Red, Black, Green
- Wears Hat
  - Yes, No

We use this set for our abstract and concrete grammar construction in the Grammatical Framework (GF) portion and semantics construction using Meta-Meta-Theory (MMT). For the inference part which actually implements the Guess Who game, we make the rules for the characters as shown in Table 1.

Name	Gender	Hair Colour	Wears Hat?
Mary	Female	Red	Yes
Peter	Male	Black	No
John	Male	Green	Yes
Fido	Other	Black	Yes

Table 1: Characters and their features for the game implementation

3.2 Grammar Trees and Semantics Construction with GL+MMT

Having built a small database for our characters and their properties, we now define the grammar to accept natural language fragments in English. First, we define the abstract grammar. This includes the categories `Name, Gender, Hair, Hat, AP`(which stands for adjective phrase), `Conn` (which stands for connectors such as and, or, etc.) and `S` (sentence) and the structure of our functions and individuals. This is of the form:

```
peter : Name;
mary : Name;
fido : Name;
john : Name;

wears_hat : Hat;
no_hat : Hat;
red_hair : Hair;
green_hair : Hair;
black_hair : Hair;
male : Gender;
female : Gender;
other : Gender;

makeAP1 : Hat -> AP;
makeAP2 : Hair -> AP;
makeAP3 : Gender -> AP;
makeS : Name -> AP -> S;
and : Conn;
combineAP : AP -> Conn -> AP -> AP;
combineS : S -> Conn -> S -> S;
```

Then, the concrete grammar provides a way to initialise types to categories and values to the functions defined above. This functionality thereby allows for different languages and sentence structures to be defined for even translation functions. The concrete grammar will look something as follows:

```
concrete GrammarEng of Grammar = {
lincat
    Name = Str;
    Hair =  Str;
    Hat = Str;
    Gender = Str;
```

```
        S = Str;
        AP = Str;
        Conn = Str;

    lin
        peter = "Peter";
        mary = "Mary";
        john = "John";
        fido = "Fido";

        wears_hat = " a hat";
        no_hat = " no hat";
        red_hair = " red hair";
        green_hair= " green hair";
        black_hair = " black hair";
        male = " male" ;
        female = " female";
        other = " other";
        and = "and";

        negateAP1 adj_hat = {fin="doesn't wear" ++ adj_hat; inf="not wear" ++ adj_hat};
        negateAP2 adj_hair = {fin="doesn't have" ++ adj_hair; inf="not have" ++ adj_hair};
        negateAP3 adj_gender = {fin="isn't" ++ adj_gender; inf="not" ++ adj_gender};
        makeAP1 adj_hat = {fin= "wears" ++ adj_hat; inf = "wear" ++ adj_hat};
        makeAP2 adj_hair = {fin = "has" ++ adj_hair ; inf = "have" ++ adj_hair};
        makeAP3 adj_gender = {fin = "is" ++ adj_gender ; inf = "are" ++ adj_gender};

        combineAP ap1 c ap2 = {fin = ap1.fin ++ c ++ ap2.fin; inf = ap1.inf ++ c ++ ap2.inf};
        makeS name ap = name ++ ap.fin;
        combineS s1 c s2 = s1 ++ c ++ s2;
}
```

This results in a grammar that accepts human language input. Thus the input "Peter wears a hat and John isn't female" results in a grammar tree as shown in Figure 2. Note that this portion of the project doesn't care for facts and only for grammar. We know from Table 1 that John isn't a female but Peter does not wear a hat but the grammar accepts the sentence, parses it and returns a grammar tree.

Once we have the grammar in place, we move onto the target logic and semantics construction, made possible by MMT. We use the concept of theories and views here. A theory is a sequence of constant declarations optionally with type declarations and definitions. Views are truth conserving mappings that transport theorems from source to target. Every time a new theory, view or grammar is specified MMT will build it.

The MMT theory is made to be isomorphic to the GF grammar and abstract syntax trees in GF are essentially terms. GF abstract grammars are essentially MMT theories and it is optimal when organised modularly. We can express semantics construction as an MMT view as seen in Figure 3 (Kohlhase et al., 2023).

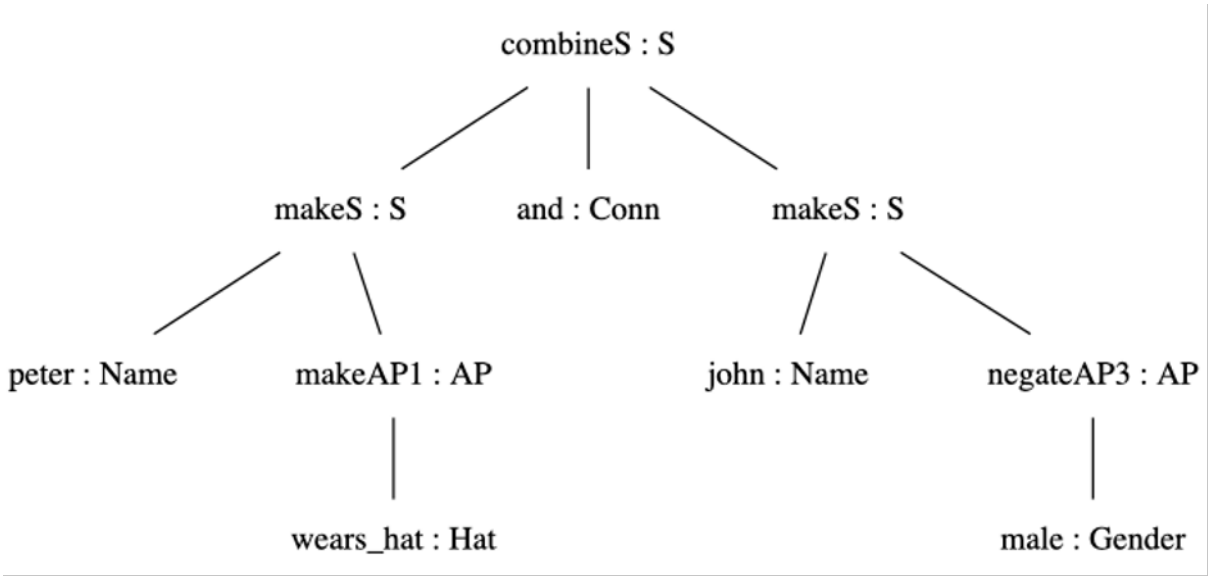

Figure 2: Grammar tree for ”Peter wears a hat and John isn’t female”

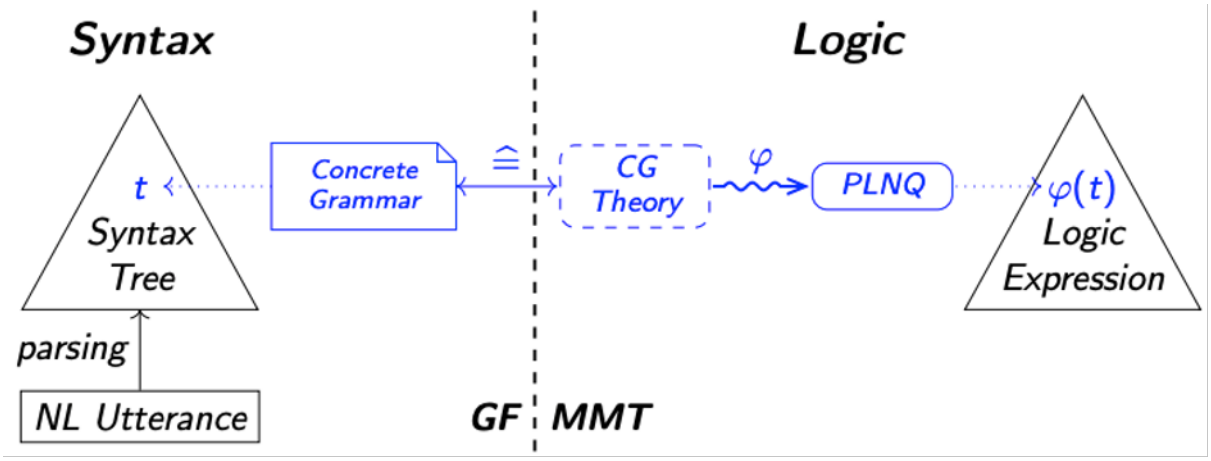

Figure 3: Conversion of syntax tree to logic through the concrete grammar theory mapping (Kohlhase et al., 2023)

To this end, we map the syntactical categories to PLNQ types, lexicon into PLNQ terms and Structural rules to defining functions via λ-terms. In the first theory `plnq`, we declare 2 types: `prop` for propositions with the short form o, `ind` for individuals with the short form ι. With this we also define the connective `and` as a $o \to o \to o$ mapping and `not` as an $o \to o$ mapping to represent how these operations are to be applied in semantics.In the second theory `people`, we define the characters' names with ι, (eg: `peter:` ι with representation `peter'`, `mary:` ι with representation `mary'`). The characteristics are also represented by the variable name followed by ', read as charateristic prime (eg: `wears_hat'`) and each have the mapping $\iota \to o$ to represent the characteristic being attributed to an individual, eg: `red_hair:` $\iota \to o$, `female:` $\iota \to o$.

We then specify the view `GrammarSemantics` that is mapping the Grammar theory on Schaefer et al. (2019) to the second theory previously specified (which already inherits the first theory plnq). Here, we can define the categories as `Name` = ι, `Gender` = $\iota \to o$, `Hair` = $\iota \to o$, `Hat` = $\iota \to o$, `AP` = $\iota \to o$, `S` = o, `Conn` = $o \to o \to o$. The remaining functions can be built using this. For example, `makeAP1` has the mapping `Hat`$\to$`AP` so we define it as `makeAP1 = [hat] hat`. `negateAP2` similarly is defined as `negateAP2 = [hair][x]`$\neg$`(hair x)` (negation of mapping `Hair`$\to$`AP`). Following the grammar, the various sentence structures and connectors are given by `makeS = [n][ap] ap n; combineAP = [ap1][c][ap2][x] c (ap1 x) (ap2 x); combineS = [s1][c][s2] c s1 s2;` `and = [a] [b] a` $\wedge$ `b .`

Now we can send semantics construction requests to the GLF-Server using the construct command. The construct command takes an abstract syntax tree as argument, thus we can adjust the command in such a way we parse a sentence and return the semantic logic expression. For example, when we pass the following command:

```
parse "Peter is male and John doesn't wear a hat" | construct
```

we get the result:

$$(male'\, peter') \wedge \neg(wears\_hat'\, john')$$

3.3 Guess Who? Inference with ELPI

ELPI implements a variant of λProlog along with Constraint Handling Rules, and thus provides a way to manipulate our abstract syntax trees with binders. It allows users to implement Higher Order Logic in their projects and can be embedded into larger applications written in OCaml as an extension language (thus representing the E in ELPI). It aims to provide a programming environment for an interactive theorem prover's elaborator component(Dunchev et al., 2015).

For our project, we define the inference rules from propositional natural deduction calculus $\mathcal{ND}_0$ for introduction and elimination of the connective `and` and `not`. These are given by

$$\frac{A \quad B}{A \wedge B} \wedge I, \quad \frac{A \wedge B}{A} \wedge E_l, \quad \frac{A \wedge B}{B} \wedge E_r, \quad \frac{[A]^1 ... C \quad [A]^1 ... \neg C}{\neg A} \neg I$$

where $\wedge I$ is the AND introduction rule, $\wedge E_l$ is the AND left elimination rule, $\wedge E_r$ is the AND right elimination rule and $\neg I$ is the NOT introduction rule. The elimination rule for NOT $\neg E$ is used only in classical logic (Kohlhase et al., 2023). Following this, we define the types first as:

```
kind proposition type.
kind proof type.
```

`proof` has been defined in such a way that it will be used to store the proof that has been evaluated and when variables with this type are called, we can access and read these proofs. From Section 3.2, we know the datatype mapping of each of sentence structures. So using this as reference, we declare the connectives and the functions to be called as:

```
type and proposition -> proposition -> proposition.
type neg proposition -> proposition.
type provable proposition -> int -> proof -> prop.
type andI proof -> proof -> proof.
type andEl proof -> proof.
type andEr proof -> proof.
```

Here, we define `provable` to act as a function that accepts a proposition and returns the truth value by virtue of the query succeeding or failing. Then we can go ahead and define the characters and the characteristics as axioms. In a similar fashion, we also define the axioms for the *and* connective and *not* sentence function.

```
type wears_hat proposition -> proposition.
type red_hair proposition -> proposition.
type green_hair proposition  -> proposition.
type black_hair proposition -> proposition.
type male proposition -> proposition.
type female proposition  -> proposition.
type other proposition -> proposition.

type peter proposition.
type mary proposition.
type fido proposition.
type john proposition.

type a_and_b proof.
type neg_a proof.
type gender proof.
type hair proof.
type hat proof.
```

Now, we have to define the actual inference rules using the `provable` axiom that we have previously defined. So we define all the information that we have (Table 1) and the introduction and elimanation rules for *and* operations. This structure of the *and* operations defined allow for different combinations of features within one sentence/proposition. Thus, the final proof can be broken down into sub-proofs and checked if it holds true as a whole. This is the reason why we define the `int` value when declaring the `provable` axiom.

```
provable (and A B) Depth (andI SubProof1 SubProof2):-
Depth > 0,
NewDepth is Depth - 1,
provable A NewDepth SubProof1,
provable B NewDepth SubProof2.

provable A Depth (andEl SubProof):-
```

```
    Depth > 0, NewDepth is Depth - 1,
    provable (and A _B) NewDepth SubProof.

provable B Depth (andEr SubProof) :-
    Depth > 0, NewDepth is Depth - 1,
    provable (and _A B) NewDepth SubProof.

provable (neg _A) _Depth neg_a .
provable (and _A _B) _Depth a_and_b .

provable (male peter) _Depth gender.
provable (male john) _Depth gender.
provable (female mary) _Depth gender.
provable (other fido) _Depth gender.

provable (black_hair peter) _Depth hair.
provable (green_hair john) _Depth hair.
provable (red_hair mary) _Depth hair.
provable (black_hair fido) _Depth hair.

% provable (no_hat peter) _Depth hat.
provable (wears_hat john) _Depth hat.
provable (wears_hat mary) _Depth hat.
provable (wears_hat fido) _Depth hat.
```

We now have an inference model based on the information we have passed to the ELPI prover in the form of axioms and inference rules. Thus, the input

```
query "provable(and(male X)(wears\_hat X)) 1 P"
```

will result in the output

$$\begin{aligned} \mathit{Query\ \ succeeded} \\ P = \mathit{andI\ \ gender\ \ hat} \\ X = \mathit{john} \end{aligned}$$

4 Challenges and Future Work

The current project works for a small subset of people and characteristics which allows for a limited scope of game play. While addition of new characters and features are easily done, it is a cumbersome task. New types of features need the creation of new data types starting at the abstract grammar level. A different data type or instruction set can be used for an integrated approach of the features.

The project can be extended to have an automatic pipeline for natural language input and implement the game in a smoother way. The handling of input such as ”Who has red hair and wears a hat?” should be able to generate the response ”Mary”. This can be even built into an application which handles the actual format of the game where a player asks ”Does your character have green hair?” and the opponent player/computer responds accordingly based on the character it chose.

The GF Framework allows for the grammars to be defined in different languages in an easy manner. Thus the game allows for an inclusive, multilingual extension that allows for a wider audience to play.

5 Conclusion

We show that given a linguistic game with clear set of rules and inference-based gameplay, a logic based system can be built and solved. In this project we use the Grammatical Framework, MMT and Embedded Lambda Prolog Interpreter. Building characters, we showed how we can use the inference models to deduce the characters and eliminate any wrong combinations. If we try to use Nica (2016)'s methods for gameplay, it can be evolved into a two-player digital game that can be played against a computer with a larger set of characters or even real images, opening up possibilities to apply and study artificial intelligence behaviours.

References

C. Dunchev, F. Guidi, C. Sacerdoti Coen, and E. Tassi. ELPI: fast, Embeddable, λProlog Interpreter. In *Proceedings of LPAR*, Suva, Fiji, Nov. 2015. URL `https://hal.inria.fr/hal-01176856`.

I. Hasbro. Guess Who Product Page, 1979. URL `https://shop.hasbro.com/de-de/product/guess-who-original-guessing-game-board-game-for-kids-ages-6-and-up-for-2-players/91A816B1-2401-4D16-BD4E-A7223F110A92`. Accessed on April 17th, 2023.

M. Kohlhase, J. F. Schaefer, M. Lattka, and N. Roux. Lecture notes in logic-based natural language processing, 02 2023. URL `https://kwarc.info/teaching/LBS/notes.pdf`.

A. Kurenkov. A Brief History of Game AI Upto AlphaGo, 2016. URL `https://www.andreykurenkov.com/writing/ai/a-brief-history-of-game-ai/`. Accessed on April 17th, 2023.

M. Nica. Optimal strategy in "Guess Who?": Beyond Binary Search. *Probability in the Engineering and Informational Sciences*, 30(4):576–592, 2016.

F. Rabe and M. Kohlhase. A scalable module system. *Information and Computation*, 230:1–54, 2013. ISSN 0890-5401. doi: https://doi.org/10.1016/j.ic.2013.06.001. URL `https://www.sciencedirect.com/science/article/pii/S0890540113000631`.

A. Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

M. Rober. Best Guess Who Strategy - 96% win record using Math, 2015. URL `https://youtu.be/FRlbNOno5VA`. Accessed on April 17th, 2023.

J. F. Schaefer, K. Amann, T. Wiesing, and M. Kohlhase. Kwarc/GLIF: The Grammatical Logical Inference Framework, 2019. URL `https://github.com/KWARC/GLIF`. Accessed on March 15th 2023.